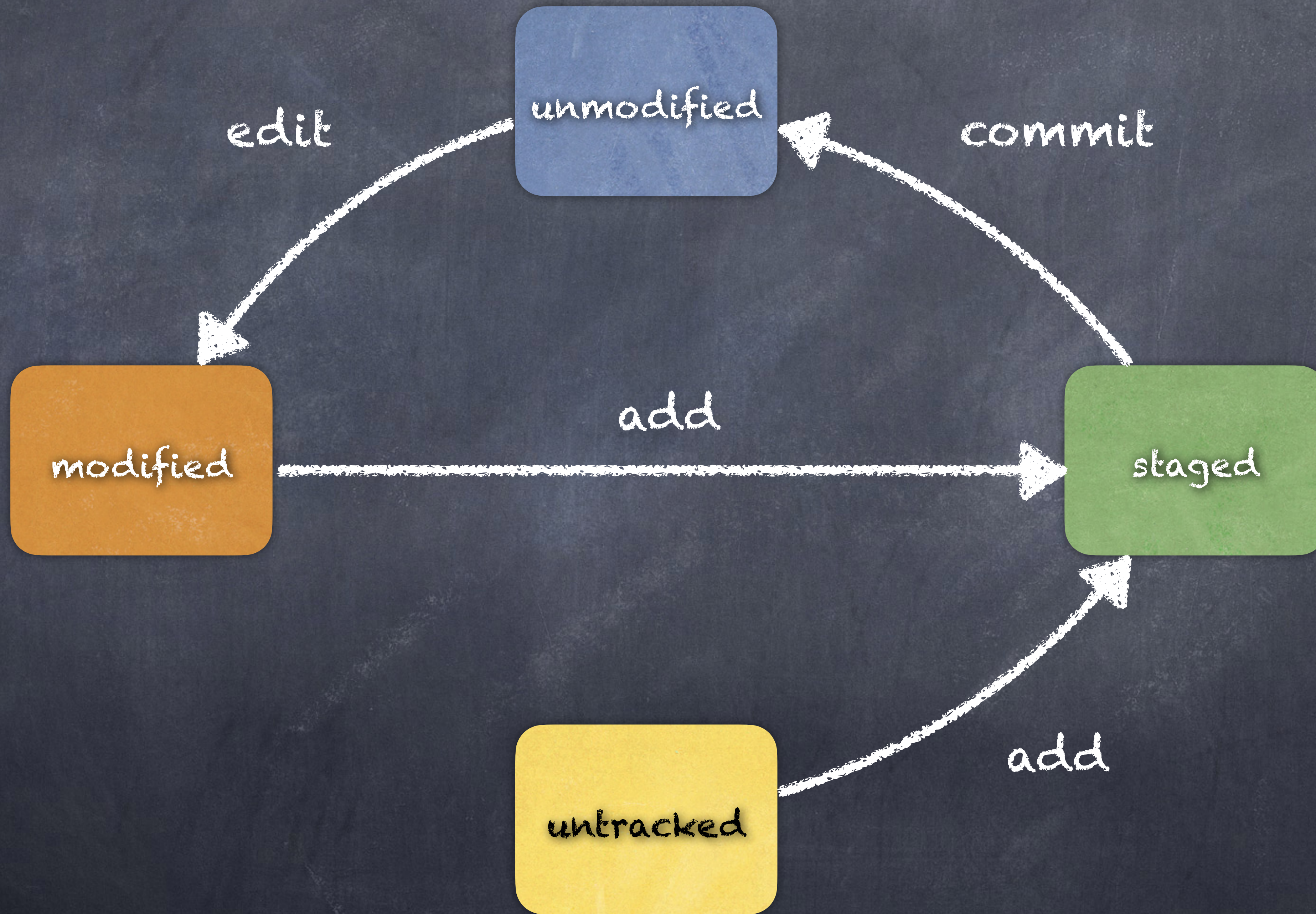
CSE306 Software Quality in Practice

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

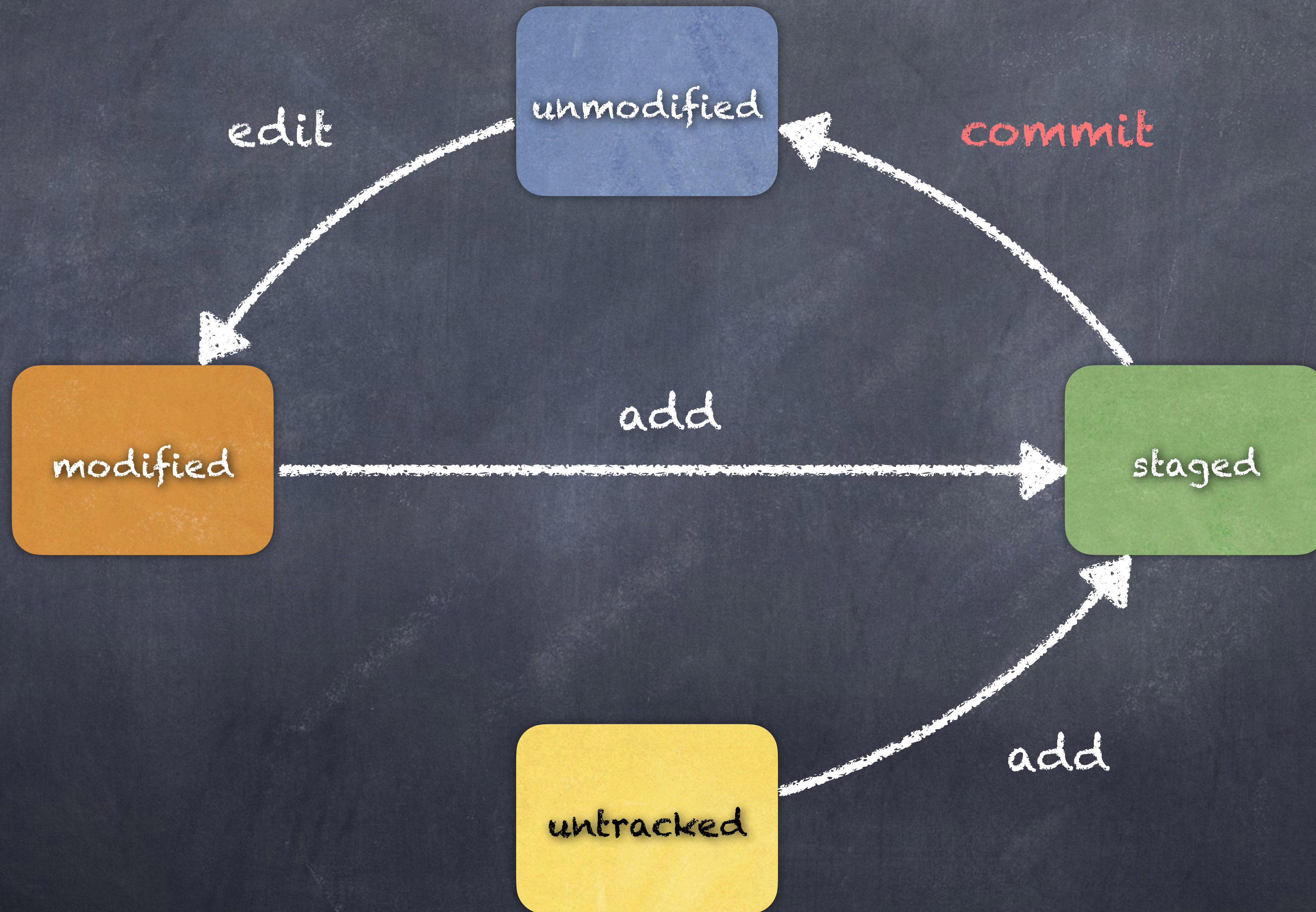
Wrapping up our intro to git

(the possible states of a file)

Possible states of a file
(git status -v)



commit preserves contents
(accidental removals can be recovered from)



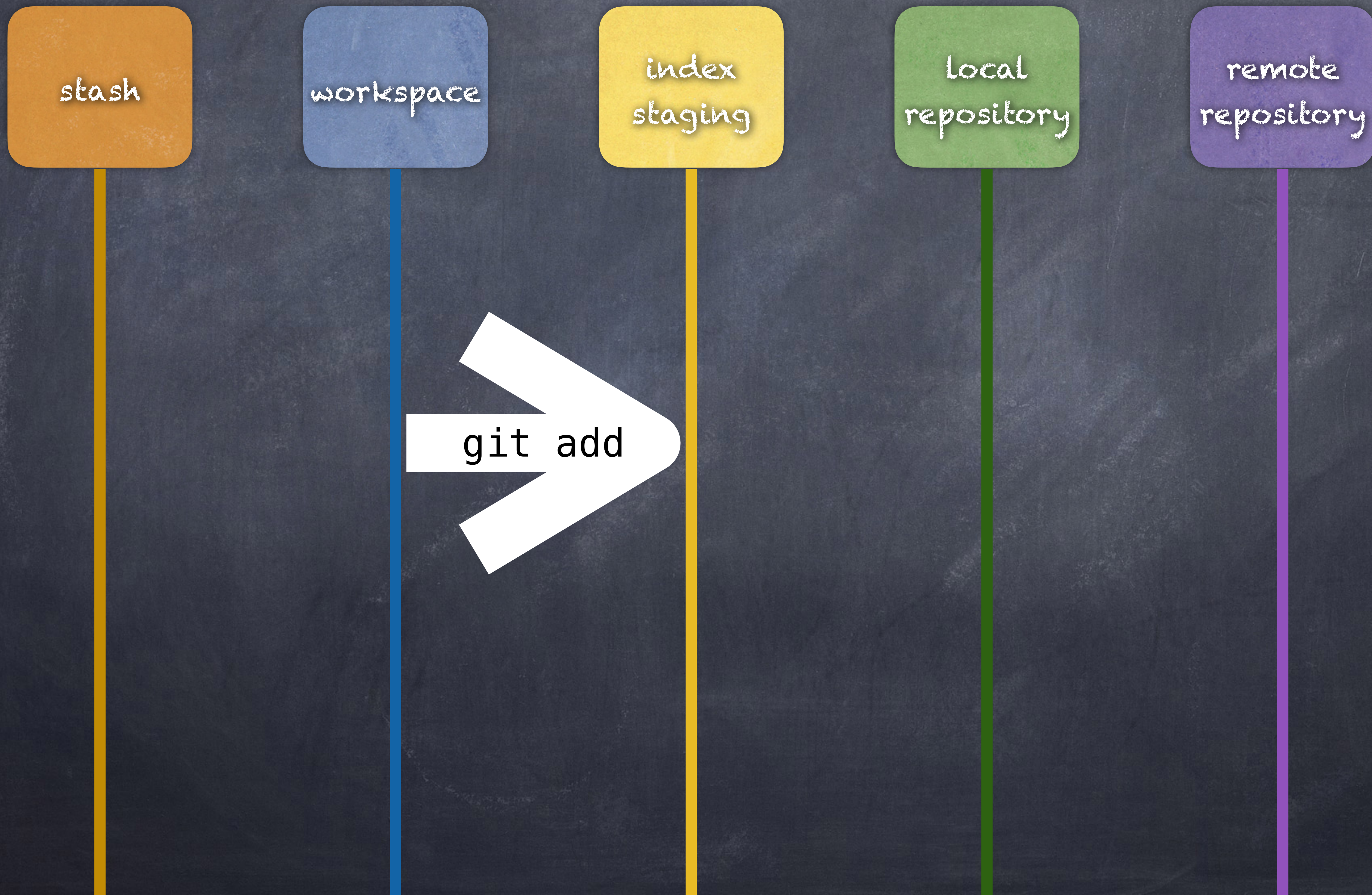
create a file

Suppose we create a file in the workspace.

How do we get it into the local repository?

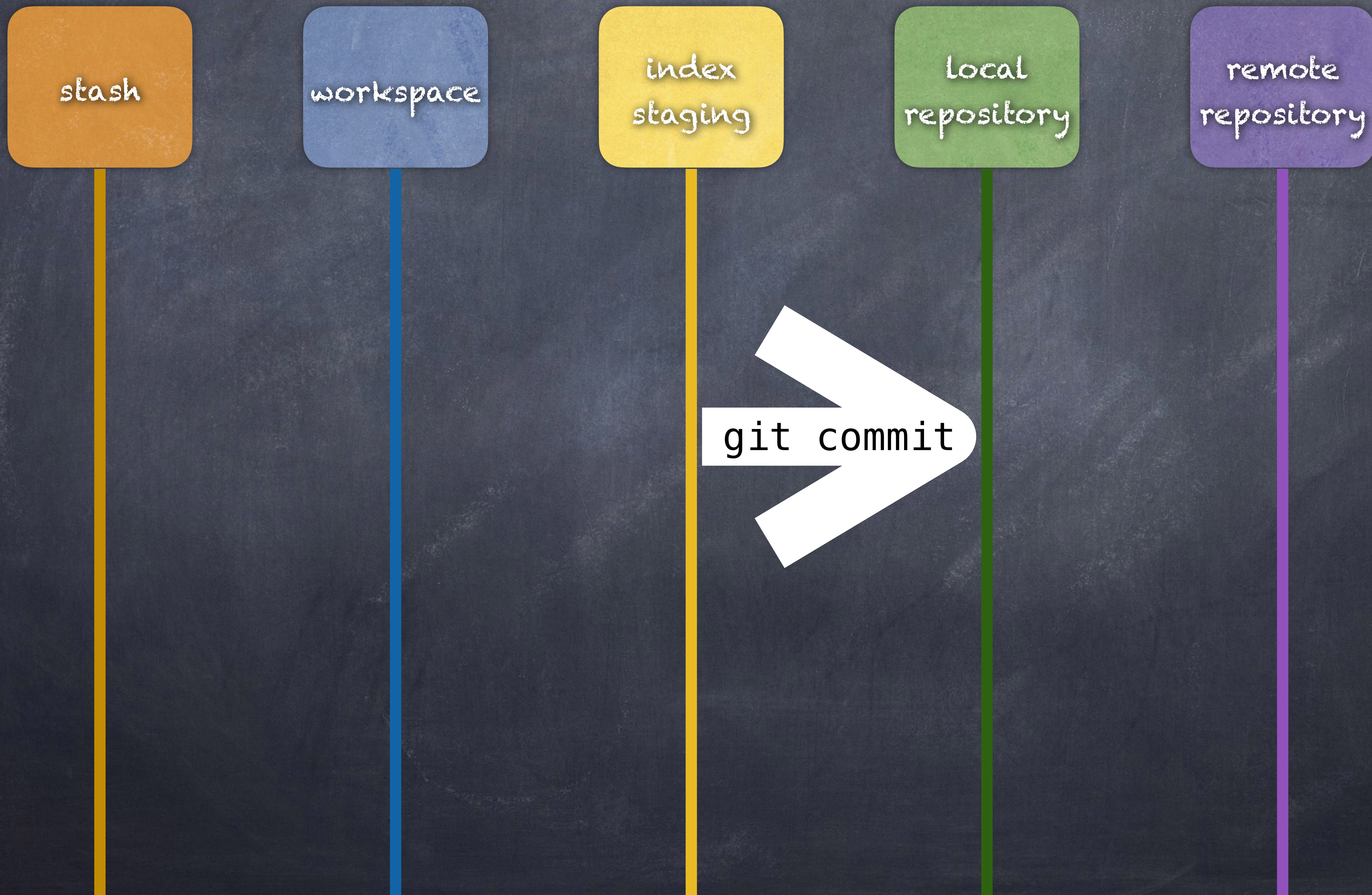
add to index (staging area)

- git add <filename>


stash
workspace
index
staging
local
repository
remote
repository
git add

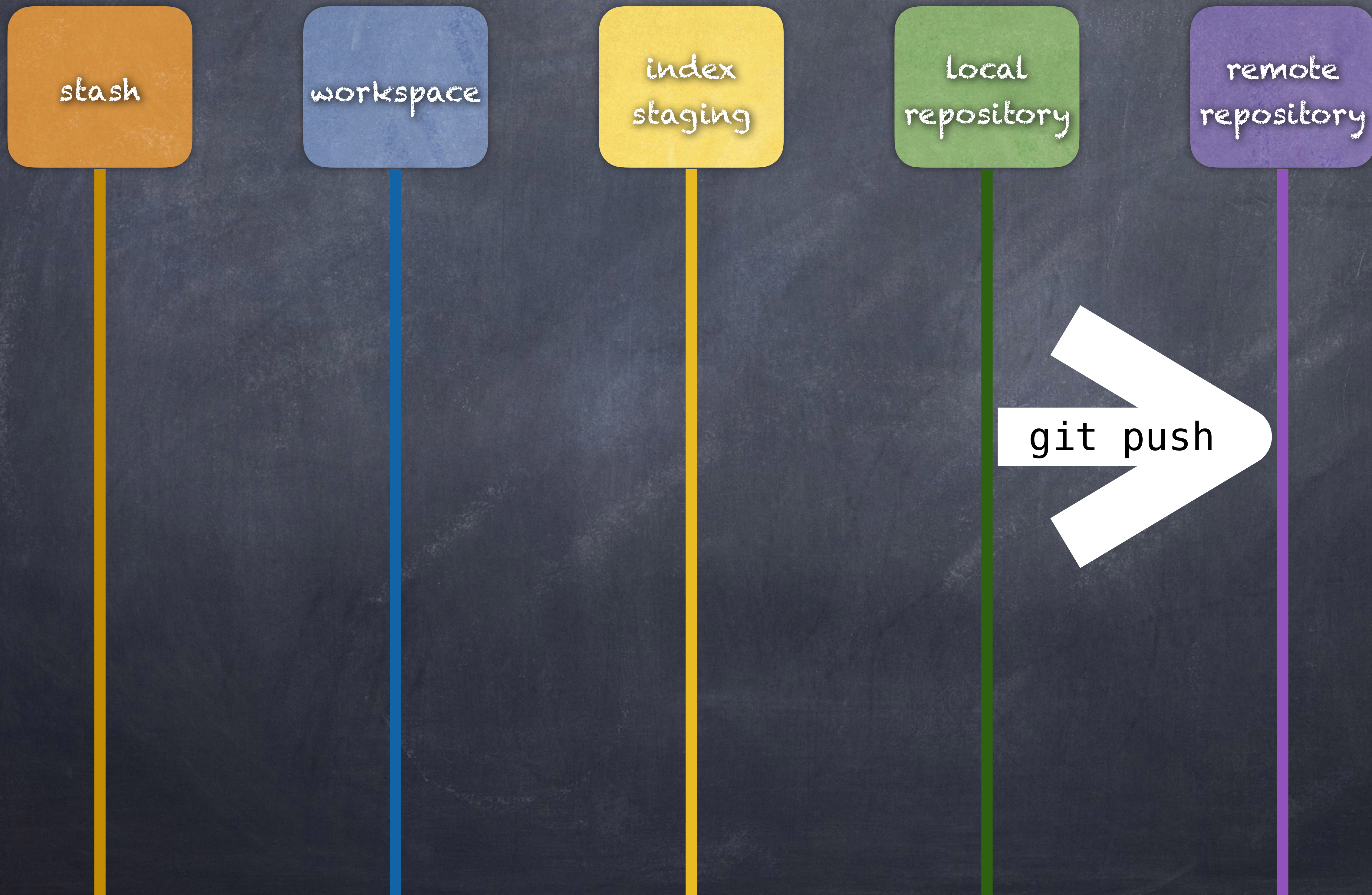
commit to local repo

- git commit -m "message"


stash
workspace
index
staging
local
repository
remote
repository
git commit

push to remote repo

- git push


stash
workspace
index
staging
local
repository
remote
repository
git push

Introduction to testing

Recall the rules

1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate fact from fiction
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didnt fix it, it aint fixed
13. Cover your bug fix with a regression test

Recall the rules

1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate fact from fiction
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didnt fix it, it aint fixed
13. Cover your bug fix with a regression test

Unit testing frameworks

- uniform way of expressing tests
- manage tests through suites
- automate testing process

Production code

Test code

Test code is separate from production code, but calls production code to verify its functionality.

Unit Testing frameworks

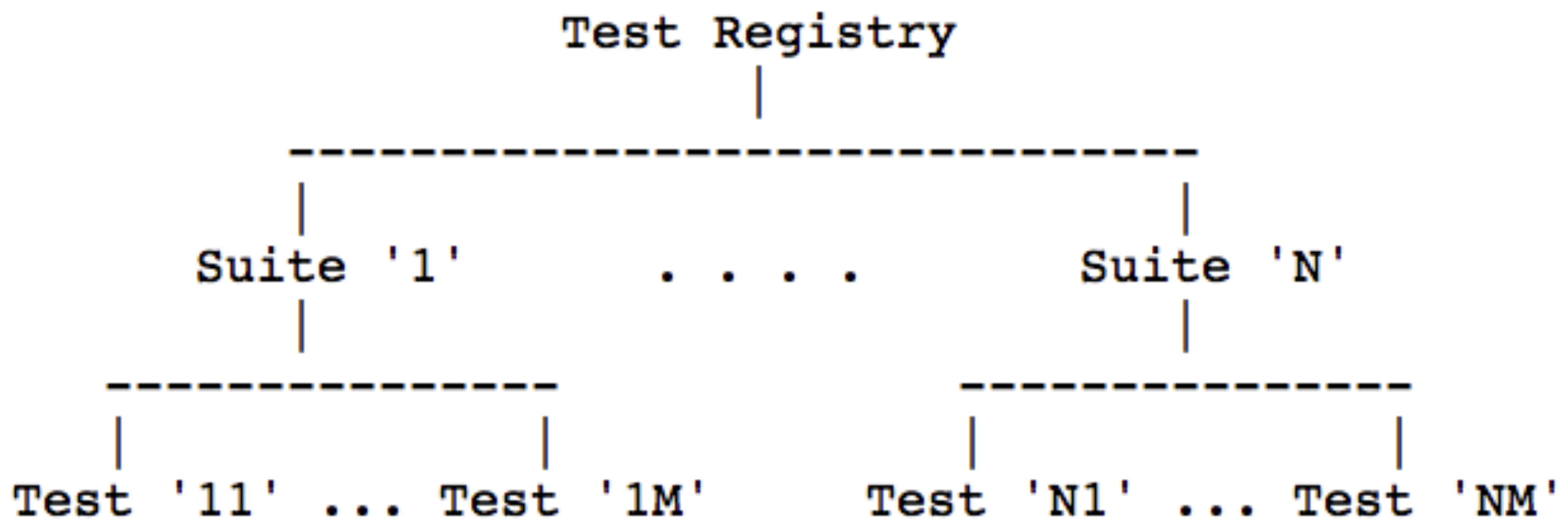
https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

CUnit - C
Criterion - C/C++
JUnit - Java
Mocha - JavaScript
pytest - Python
ScUnit - Scala
VUnit - Verilog/VHDL
among many, many others

We'll use Criterion

https://criterion.readthedocs.io

https://github.com/Snaipe/Criterion



http://cunit.sourceforge.net/doc/introduction.html

- Tests are automatically registered when declared.
- Implements a xUnit framework structure.
- A default entry point is provided, no need to declare a main unless you want to do special handling.
- Test are isolated in their own process, crashes and signals can be reported and tested.

From: https://github.com/Snaipe/Criterion

https://github.com/Snaipe/Criterion#readme

Assertions
(the most common ones)

https://criterion.readthedocs.io/en/master/assert_old.html

https://criterion.readthedocs.io/en/master/assert.html

Exercise

Write tests for a function named eval which takes two int values (x and y) and returns their sum as an int.

code from lecture

code.h

```
int eval(int,int);
```

A function prototype (i.e. a function declaration)

code.c

```
#include "code.h"

int eval(int a,int b) {
    return 0;
}
```

A stubbed out implementation (i.e. a function definition)

code from lecture

tests.c

```
#include <criterion/criterion.h>
#include "code.h"

Test(sum, test_0) {
    int x = 2;
    int y = 2;
    int expected = 4;
    int actual = eval(x,y);
    cr_assert_eq(actual, expected,
                 "I expected eval(%d,%d) to be %d, but it was %d.\n",
                 x, y, expected, actual);
}

Test(sum, test_1) {
    int x = -2;
    int y = 3;
    int expected = 1;
    int actual = eval(x,y);
    cr_assert_eq(actual, expected,
                 "I expected eval(%d,%d) to be %d, but it was %d.\n",
                 x, y, expected, actual);
}
```

Two tests functions. Each is self-contained (parameters, no value returned).

x and y denote the inputs.
expected is the expected correct value.
actual is computed by calling the function under test.

The cr_assert_eq call checks the result of the test.

Running the tests

```
turing:~/CSE306/code/UnitTesting> ./tests
[----] tests.c:9: Assertion Failed
[----]
[----]   I expected eval(2,2) to be 4, but it was 0.
[----]
[----] tests.c:19: Assertion Failed
[----]
[----]   I expected eval(-2,3) to be 1, but it was 0.
[----]
[FAIL] sum::test_0: (0.00s)
[FAIL] sum::test_1: (0.00s)
[====] Synthesis: Tested: 2 | Passing: 0 | Failing: 2 |
Crashing: 0
turing:~/CSE306/code/UnitTesting>
```

code from lecture

```
#include <criterion/criterion.h>
#include "code.h"

Test(sum, test_0) {
    int x = 2;
    int y = 2;
    int expected = 4;
    int actual = eval(x,y);
    cr_assert_eq(actual, expected,
                 "I expected eval(%d,%d) to be %d, but it was %d.\n",
                 x, y, expected, actual);
}

Test(sum, test_1) {
    int x = -2;
    int y = 3;
    int expected = 1;
    int actual = eval(x,y);
    cr_assert_eq(actual, expected,
                 "I expected eval(%d,%d) to be %d, but it was %d.\n",
                 x, y, expected, actual);
}
```